**汇编语言大作业**

**C语言的秘密**

1303105 班

刘洋

学号1133710102

2014年11月19日

目录

引言5

1. 比较C语言与ASM要素5

2. 深入阐述c语言实现5

第一章 简要描述对应关系5

1. 常量5

2. 不同类型变量5

| 一般数据类型 5

| 指针 5

| 字符串 6

| 多维数组 6

| 结构体 6

| 类 6

3． 计算常、变量表达式7

| 简单计算 7

| 位运算 7

4． 宏7

5． 函数，子程序8

6． 其他结构的汇编对应8

第二章 深入阐述C语言的实现8

1. 内存的使用8

| 程序头 8

| 代码段 8

| 部分无法用立即数表示的常量8

| 声明注释 9

| 全局变量，静态变量 9

| 需求文件，函数，链接库等 9

| 栈 10

2． 类型10

 全局变量和局部变量的不同分配 10

 普通变量11

 字符串11

 多维数组11

 指针 11

 结构体 11

 类 12

3． 指针14

 存储方式14

 指针取值（\*p） 14

 指针传递 14

4． 函数14

 确保堆栈平衡（上层函数）14

 传参数（上层函数） 15

 中间层15

 函数保护主ebp15

 堆栈让出子程序变量空间16

 执行函数体16

 准备返回值16

5． 堆栈框架18

6． 全局、局部变量19

 全局变量19

 局部变量19

7． 算数运算及表达式计算19

 常量表达式20

 简单变量表达式计算20

 需要存储中间变量的表达式计算20

8． C语言编译、连接、装入内存21

 编译21

 链接21

 装入内存21

9． C语言IO功能实现21

第三章 c语言与汇编语言优缺点及适应场合21

1． C语言优点，汇编语言的不足21

2． 汇编语言优点，c语言的不足21

3． c语言使用场合22

4． 汇编语言使用场合22

第四章 课程收获及想法22

1． 收获及想法22

2． 一些建议23

引言

1. 比较C语言与ASM要素

具体比较内容为：各种类型-含字符串及多维数组、指针、结构联合-可嵌套、类-可继承等，常量变量及表达式，算术及逻辑运算，宏与子程序-函数，全局变量、局部变量-含静态变量，参数及传递-传值或地址、返回值等。

1. 深入阐述c语言实现

具体阐述内容为：类型、指针、函数、堆栈框架、参数、全局、局部变量，类的private、public域及成员函数及以及IO功能的实现，结合有符号、无符号数及其运算，表达式与宏等论述C程序到底如何编译、连接、装入内存以及与执行的。

第一章 简要描述对应关系

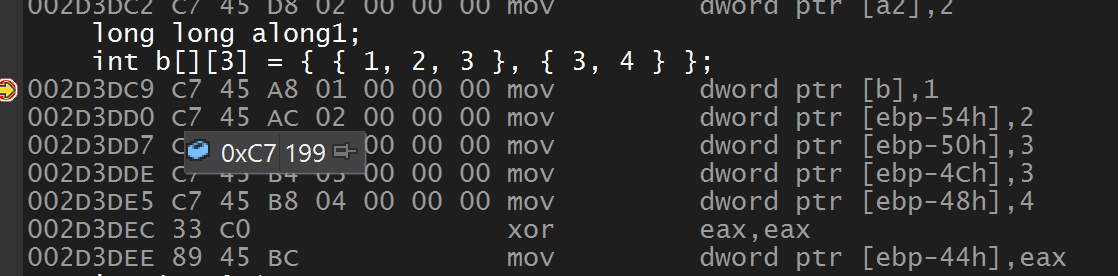
1. 常量

常量如果是数，那么会被转化成立即数。而字符串常量会保存在一块固定的内存中，等使用时复制过来，或直接使用其地址。具体会在第二章内存使用中详细介绍。

1. 不同类型变量

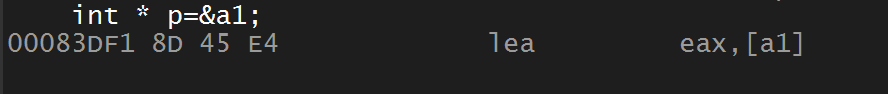
* 一般数据类型

C语言中对于变量的声明并没有翻译成汇编命令，对于没有初始化的变量，不对应汇编指令，对于初始化的变量，也没有声明语句，只是把变量赋值。



* 指针

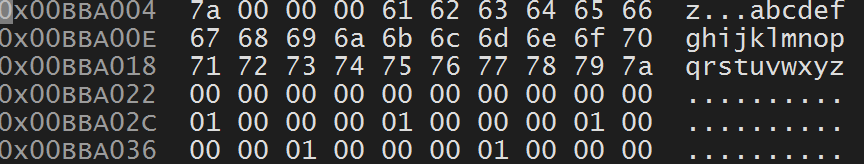
指针在c语言中被编译成了一个4字节的变量，这个变量存放的是地址值。指针指向一个变量的地址时，使用的是lea命令。





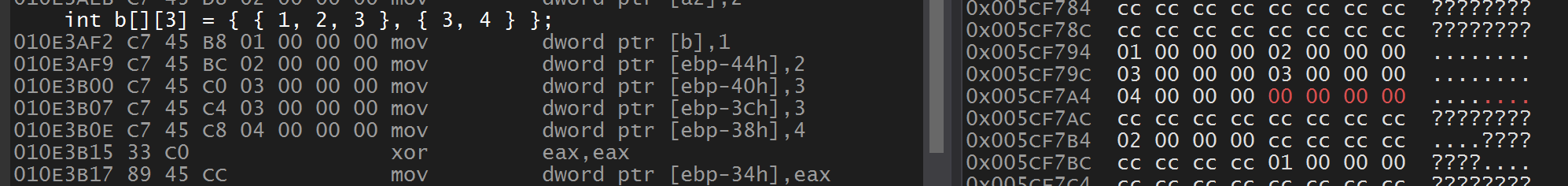
* 字符串

c语言中的字符串是一个字符数组，数组中存放着字符，并在最后补\0，常量字符串在复制时采用寄存器暂存的方式复制。



* 多维数组

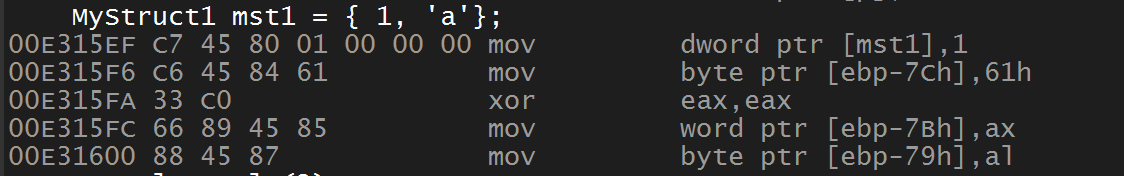
c语言中多维数组实际上是一段维数相乘大小的一维数组。对应内存一维数组（各维度的积）\*sizeof(type)的区域。初始化时没有初始化的会被附0 。



* 结构体

结构体是一种类型定义，定义并不对应指令。

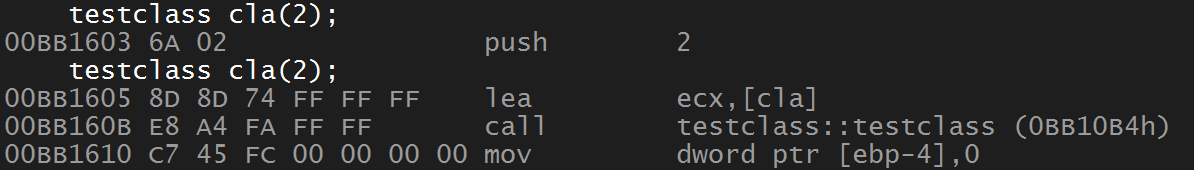
结构体在c语言中被处理成连续的多个变量，在函数中调用的时候会使用一个对ebp的偏移量进行不同字段的检查。



* 类

类亦是一种类型，定义并不对应指令。

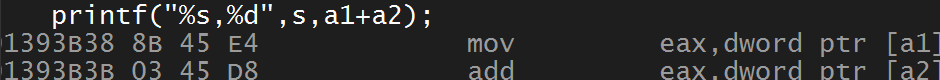
对象的声明时会调用类对象的构造函数，作用域结束的时候执行析构函数。每次调用类的函数时初始化ecx为类的this指针，用此方法来区分不同的类。



1. 计算常、变量表达式

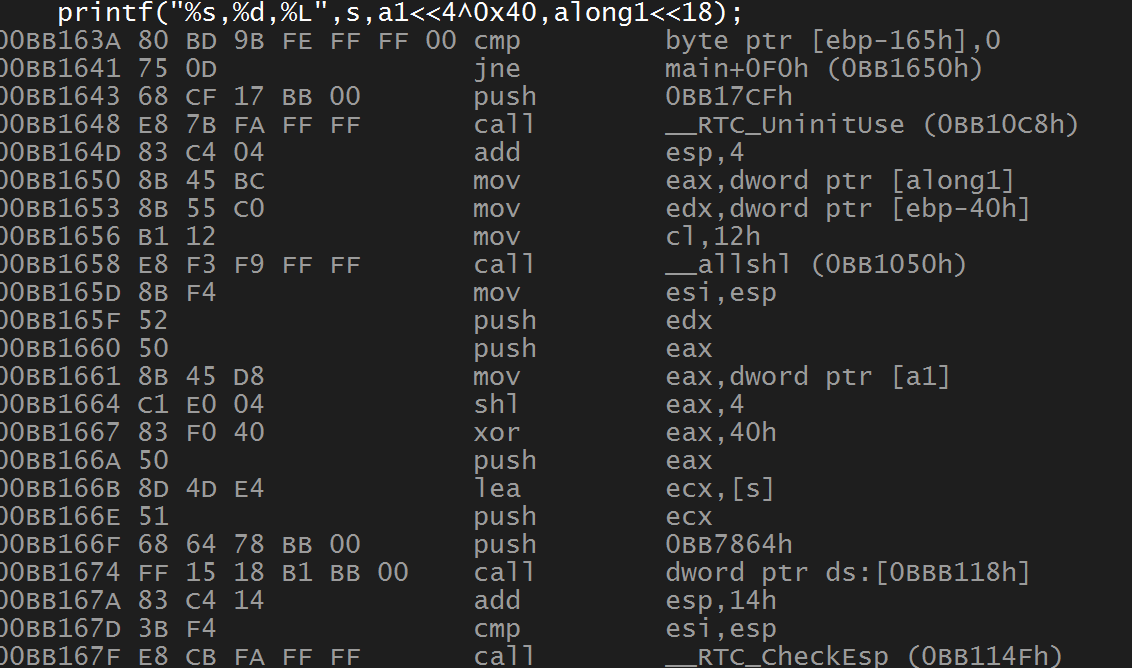
* 简单计算

常量表达式被编译器计算成立即数，变量表达式中的常量运算被编译器在编译时计算出来，表达式中变量的计算被编译成对于eax，ecx，edx，esi等寄存器的运算，包括mov，add，sub，imul，idiv等。表达式计算会在之后的第二章深入阐述中详细描述。



* 位运算

c语言翻译出的位运算是使用shl xor and or等进行移位、异或、与、或运算的。特别的，对于long 或 longlong等长于4字节的数据，移位运算使用函数，如下图中的\_\_allshl等进行运算。

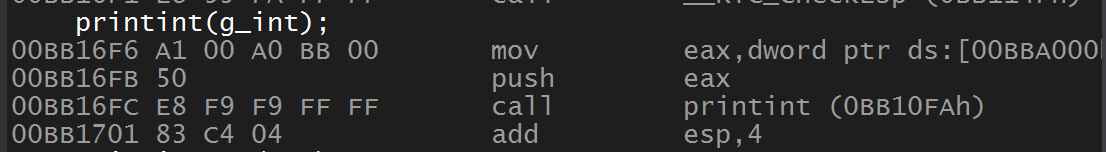


1. 宏

宏在编译前被替换之后再进行编译，宏定义存在多种情况，也会被编译为多种情况。

条件编译，c语言的宏可以包含类似#ifdef #else #endif 等条件编译指令，这些指令嵌套的c语言内容可以只编译分支中的代码，这解决了重定义和跨平台的问题。

1. 函数，子程序

函数与子程序都会被写成汇编中的子程序，使用call方式调用。

1. 其他结构的汇编对应

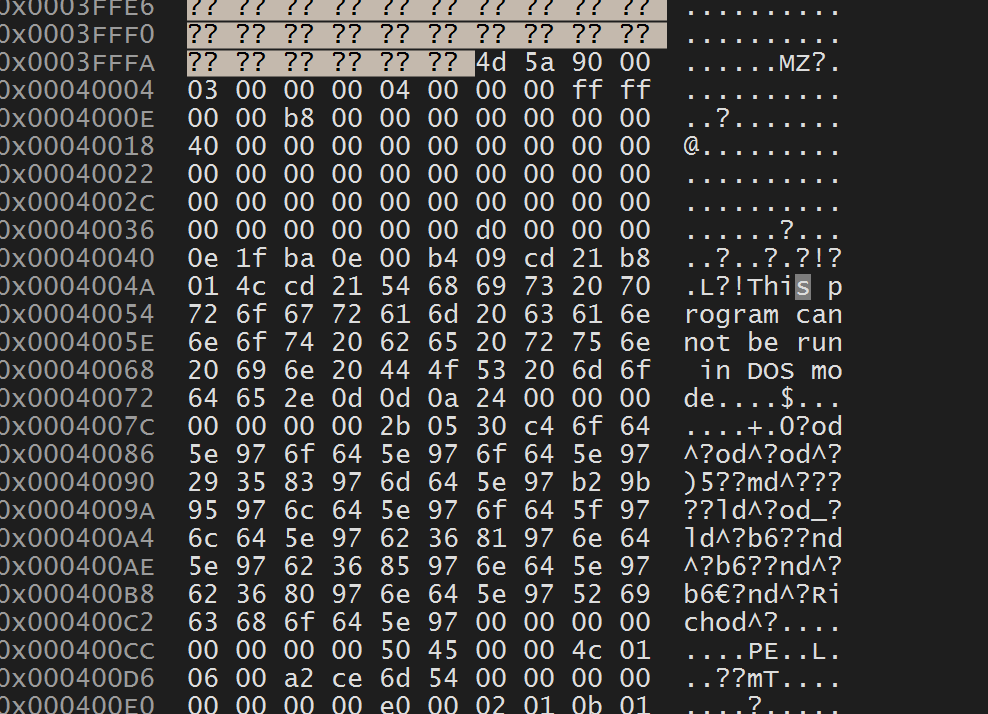
由于其他结构设计内存方面较多，甚至有些并不能简单阐述，所以将有关“全局变量、局部变量-含静态变量，参数及传递-传值或地址、返回值等”的汇编对应放在第二章中相关部分一起说明。

第二章 深入阐述C语言的实现

1. 内存的使用

一个程序在内存中是这样分配，由上到下依次是内存低地址到内存高地址。每段存储空间中都有大量以0填充的无意义字节分隔。

* 程序头

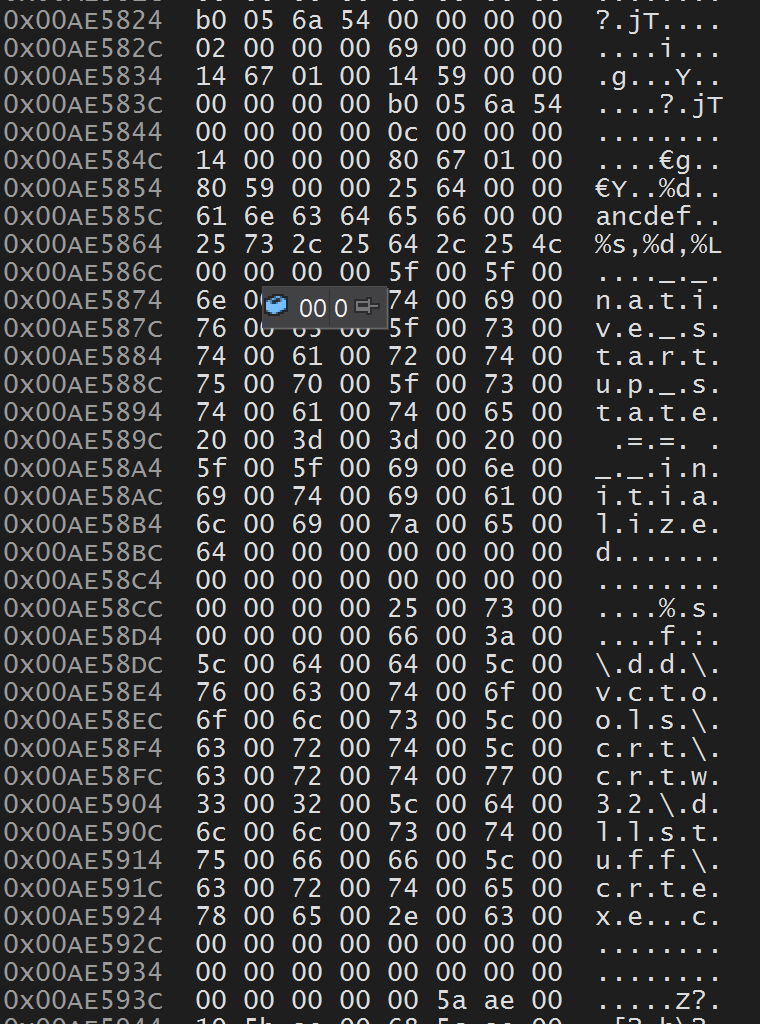


* 代码段

在加载如内存的低地址的位置存放着代码（指令）。

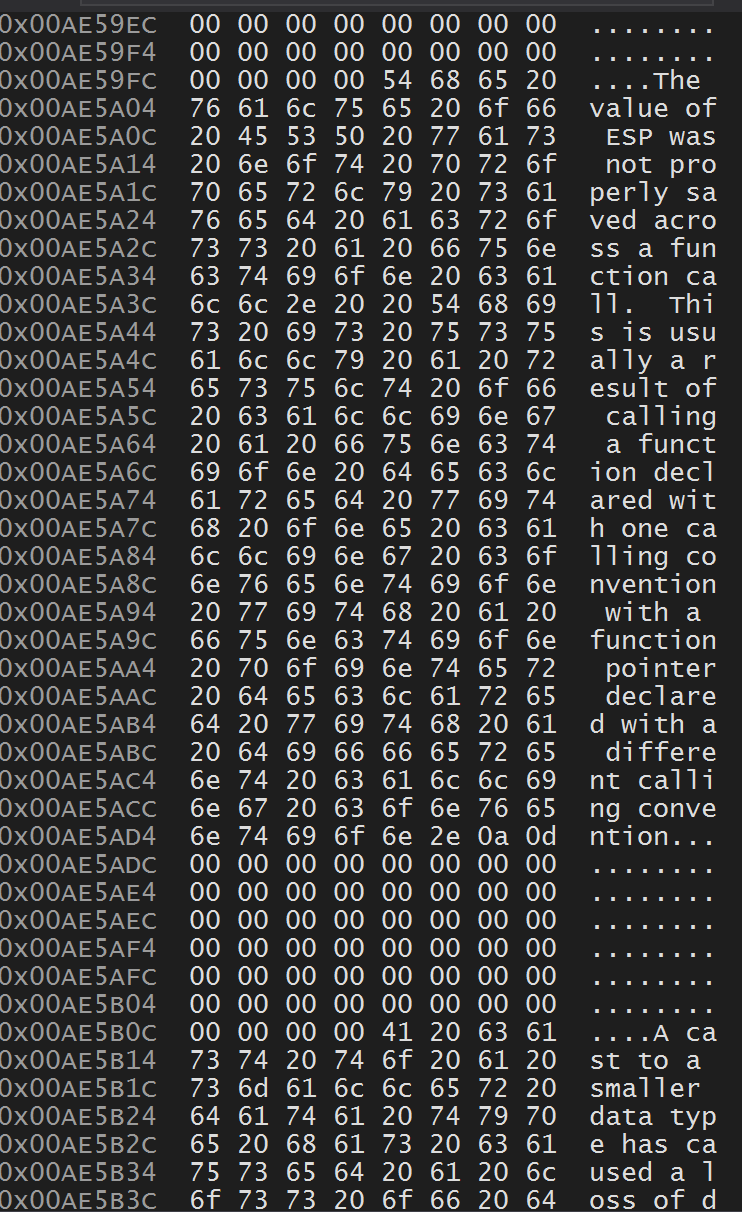
* 部分无法用立即数表示的常量

此处存放的是如printf使用的常量字符串，初始化使用的常量字符串等等。这些字符串是相邻存放的，但是他们都对齐了4字节，没有对齐的用0补齐。如字符串”abcdef”字符串加上最后的‘\0’一共占7字符，系统增加一个0补齐4的倍数。



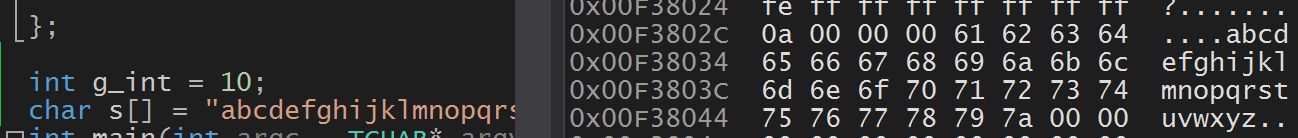
* 声明注释

再向下保存了一些程序的声明，比如esp使用方式、Microsoft的一些程序定义，以及工程位置等信息。

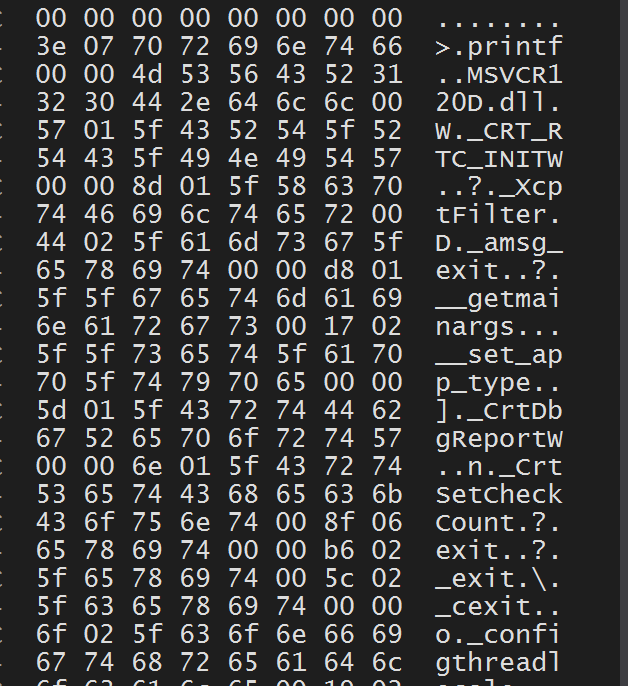


* 全局变量，静态变量

全局变量是指C语言中函数外声明的变量。这种变量的作用域为文件域。特别的，c语言中函数中的静态变量实际上在实现上是和全局变量不做区分的。

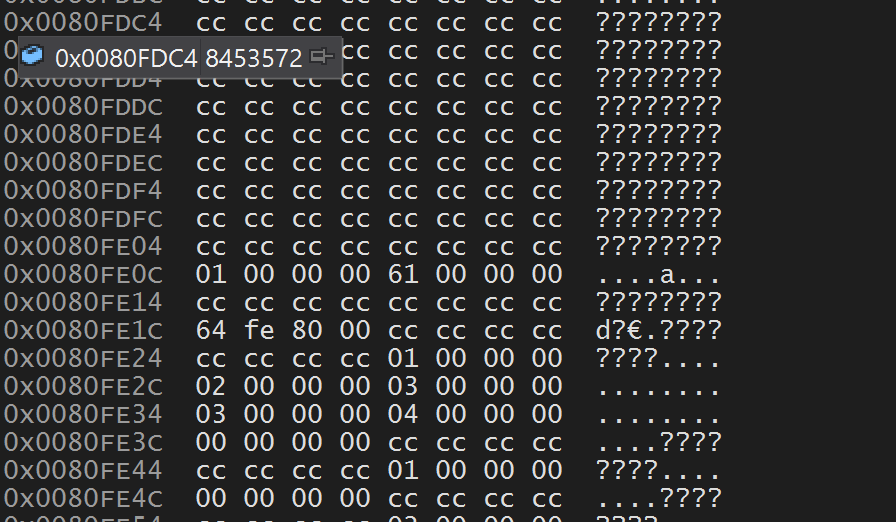


* 需求文件，函数，链接库等



* 栈

栈结构是C语言程序中最主要的存储位置。进入主函数后几乎所有变量都存在栈中。

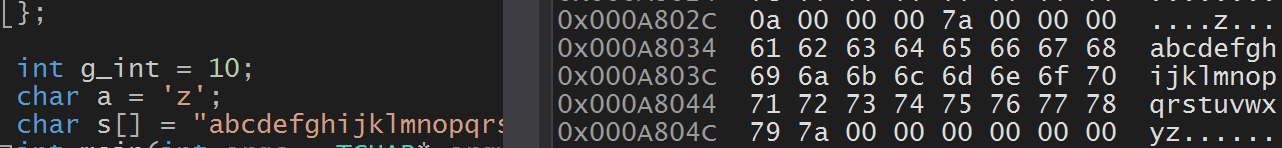


1. 类型

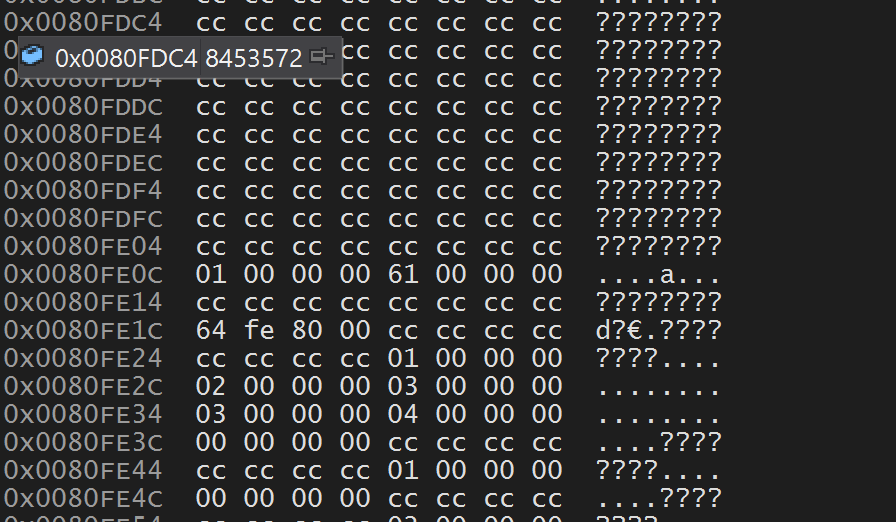
* 全局变量、静态变量和局部变量

全局变量和静态变量是保存在上节提到的全局变量保存空间内的，是连续存储的。而局部变量存储在堆栈段，中间相隔8字节以cc填充的空间。

下图展示的是全局变量的存储方式，存储在堆栈段，每个变量需要对齐4字节，如字符’z’之后被填上3字节0，对齐内存地址为4的倍数。对其之后的变量顺序摆放，并没有间隔。



下图展示的是堆栈端内局部变量分配时的补齐和间隔。可见，每个变量都对齐4字节，没对齐的用0补齐，每个变量之间相隔8个字节的cc，这个cc是进入函数时预先填入的，详见函数一节。



* 普通变量

普通变量对应一个地址，使用时对应[ptr]，初始化使用立即数，计算时使用eax保存，超过4位另使用其他寄存器操作。

* 指针

指针在c语言中被编译成了一个4字节的变量，这个变量存放的是地址值。具体会在本章指针一节详细介绍。

* 结构体

结构体声明不会翻译成指令，只是在调用和初始化的时候，编译器自动翻译成相应的地址偏移，如下图，我们定义的结构体如下。

struct MyStruct2

{

char c1;

char c2;

};

struct MyStruct1

{

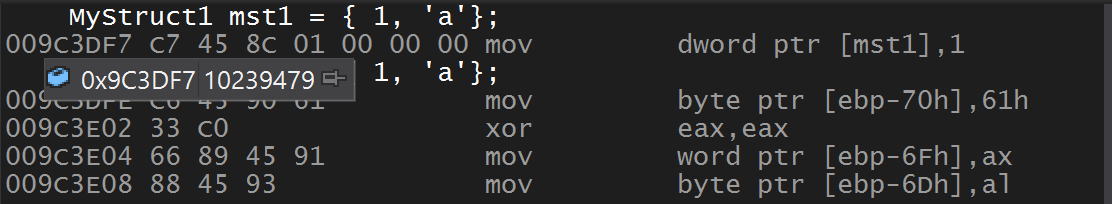
int int1;

char c;

MyStruct2 x;

};

如下图，可见一个结构体被翻译成如此5句指令。这5条指令都是用于初始化的。在结构体0偏移位置，置1 。给第二个元素c赋值‘a’既61h，这里可以看出，给其他字段赋值时使用的是相对ebp的偏移，C语言中保留结构体形式，实为另一个局部变量。之后会给没有赋值的其他字段负0，这里采用的方法是，使用xor eax，eax命令将eax置零，再写入其他字段中。



* 类

类的对象对应一块独立的存储区域和一块共享的函数代码，特别的static变量使用的是一块公用的存储区域。所有类的对象都使用一块代码（函数），而使用ecx保存的this指针指向的数据段不同，来区分不同的对象。

类的初始化

类的定义为

class testclass

{

public:

testclass();

~testclass();

testclass(int a);

private:

int data;

int data2;

};

testclass::testclass(int a)

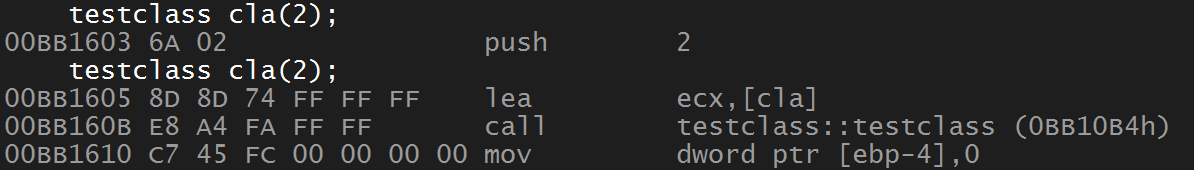
{

data = a;

data2 = a\*2;

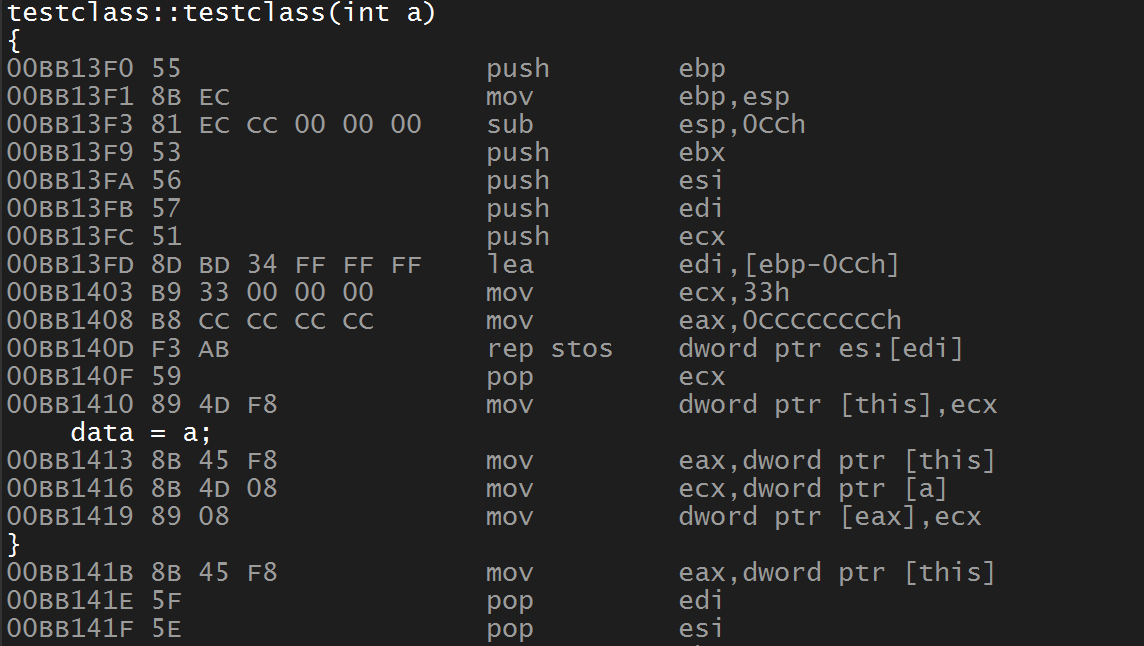
}

对象的定义和构造如下图。



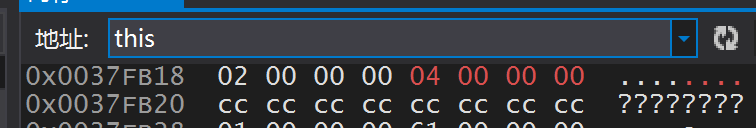
我们看出，这个构造函数调用除了按照普通函数调用以外，还初始化了ecx，将ecx置为对象cla的地址，既c++中的this指针。

如下图进入构造函数中我们看出。函数要使用ecx进行腾出栈位置并写入cch的操作，故在操作时用栈保存了ecx。在函数开始的最后一句，将ecx的值存入this变量中，this变量并不是类中显式声明的变量，而是对象中本身具备的this指针域。



this指针指向的是对象的数据域的第一个元素

如下图，向构造函数中传入2，调用时将this指向内存开始位置置为2，之后再有字段会向后延续排列。下图正在初始化类中的第二个字段data2 。



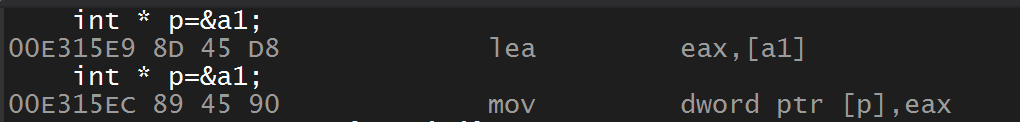
下面我们讨论类的成员权限问题，对于一个类的共有（public）、私有（private）、保护（protected）的字段和函数，在字段在内存上的保存和字段的操作上并没有区别，private的字段会和public字段都存储在对象this指针指向的连续内存区域内，调用和赋值时并没有区别。

这些权限只是编译上的限制，比如在类外调用类内部私有成员函数和属性时是无法通过编译的，这是C++或者说面向对象编程所限制的，能够使得程序结构化，提高函数（操作）、数据的封装性而提出的概念，在实现上并没有特殊的区别。

1. 指针

* 存储方式

指针在c语言中被编译成了一个4字节的变量，这个变量存放的是地址值。符合本章4节描述的在函数中变量的8字节间隔，对齐等要求，可以理解为就是一个4字节长度的变量。



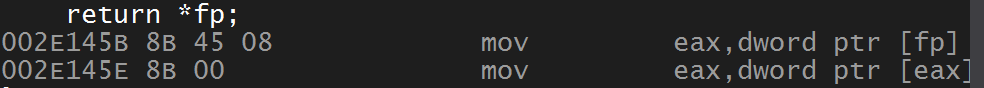
* 指针取值（\*p）

指针取值一般被编译为两条指令，类似

mov eax,dword ptr [p]

mov eax,dword ptr [eax]

先把指针赋给eax再取一次eax这个地址上的值，这时eax即为\*p。



* 指针传递

函数传入指针时，实为传入一个4字节的数（地址值），虽然叫按地址调用，但是也进行了传递参数的过程，只不过这个参数是一个地址值，在函数中，通过取这个地址的值和修改这个地址的值对这个地址上原有的变量进行操作。因为也进行了传值过程，这个函数内部修改这个地址只针的指向并不能改变主函数中传入的那个指针变量在主函数中的值。

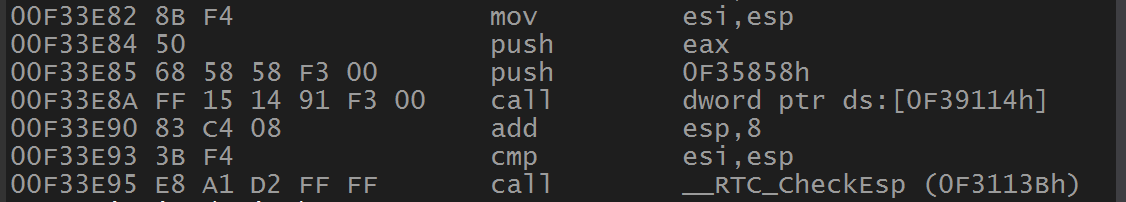
1. 函数

调用函数C语言编译器编译的指令会顺序执行如下操作

* 确保堆栈平衡（上层函数）

在调用之前的第一步需要储存esp的位置，保证函数调用结束之后堆栈恢复原始位置。

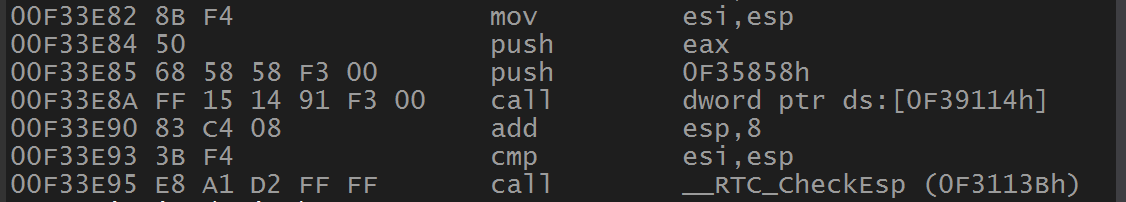
使用的方法是，在esi中保存esp（mov esi,esp），并在函数调用结束后比较esi,esp查看其是否还相等（cmp esi,esp）即可判断堆栈（esp）是否和调用前一致。这个判断和异常处理调用另一个函数\_\_RTC\_CheckEsp中执行。由于esi在进入函数时会被压栈，并不会改变，如果esp位置改变，pop出的esi再与esp相等是小概率事件，使用这种方法很难出错。



* 传参数（上层函数）

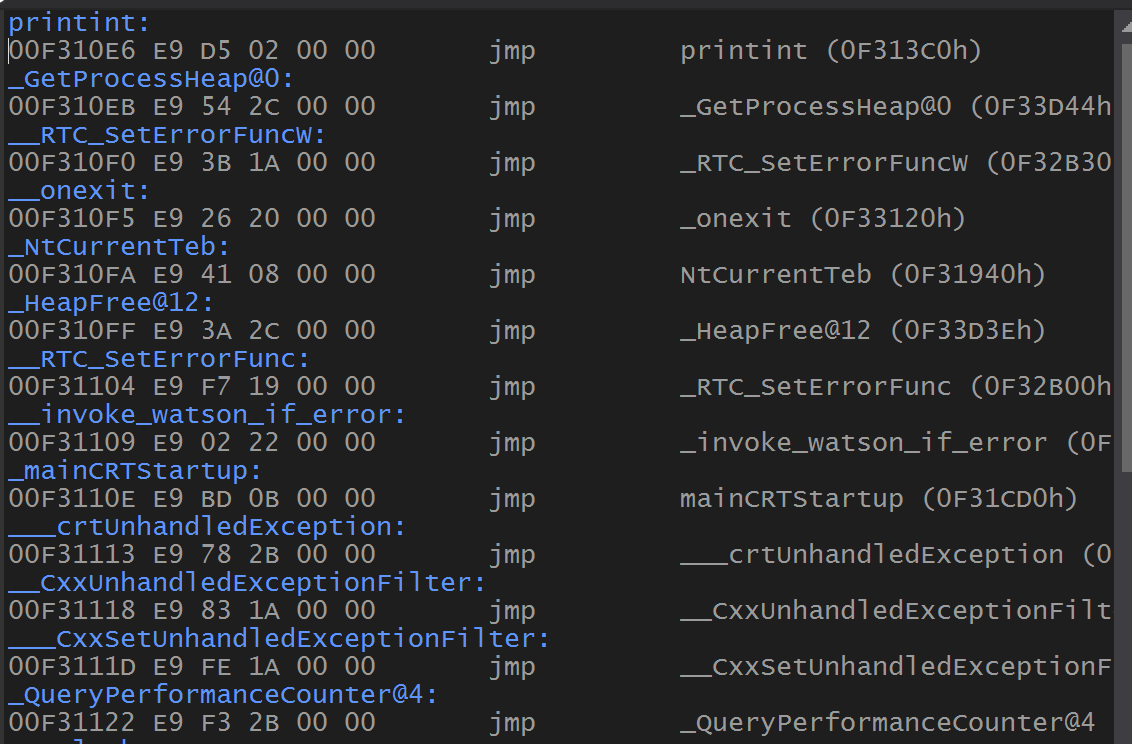
首先准备传入的参数，使用push指令压栈进行传递，压栈顺序是按照参数在列表中的排列，从后向前压栈。

之后，使用call指令进行调用子函数。



* 中间层

在主函数调用函数并不会直接跳到子函数中，还会经过一个中间层，在中间层中保存着程序用到的所有函数的地址，相当于一层保护，防止程序任意跳转。（下图为跳入中间层）



* 函数保护主ebp

调用call后自动会将返回地址入栈，在返回地址上一个4字节内存中，会压入ebp。

之后使用ebp存储esp（mov ebp，esp）。

* 堆栈让出子程序变量空间

使esp减一个数（我们暂且成为x，x为8的倍数），让堆栈腾出一定的空间，并将现在的esp存入edi中。

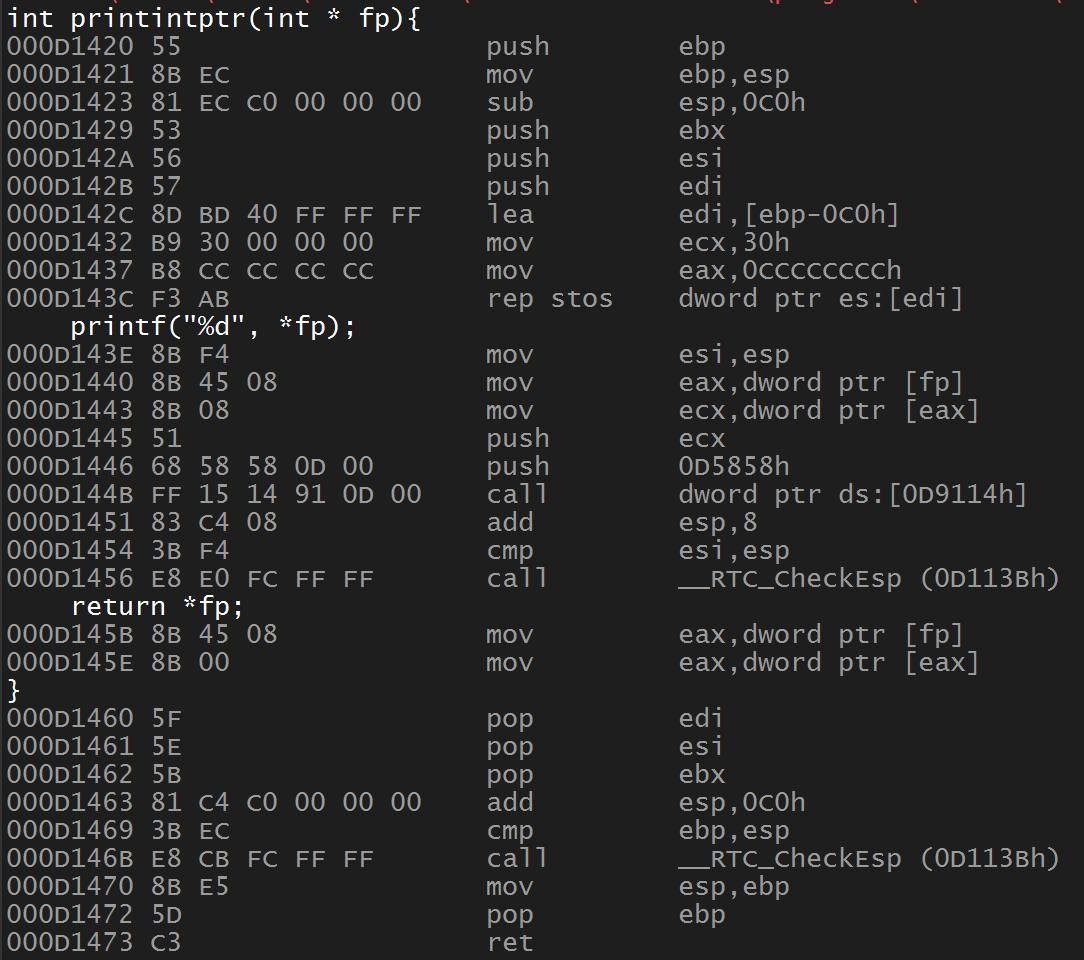
之后将这个空间进行初始化，这个初始化过程将ebp到esp之间的内存每一位置为0CCh。

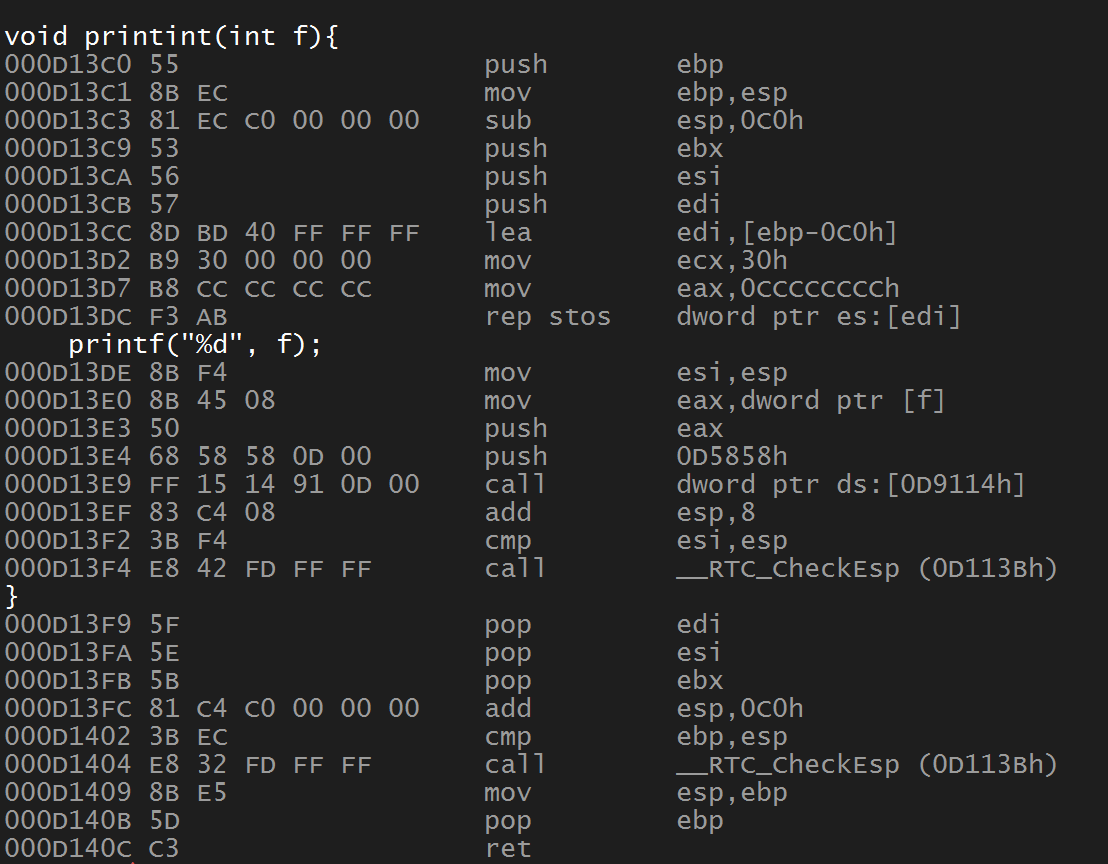
实现方式为先将eax置为0CCCCCCCCh，置ecx为（x/4），因为eax为4字节，每次写内存可以置位4字节。使用rep stos 命令将eax重复置位到这x个字节中，既得到x个字节的cc cc … cc cc 。



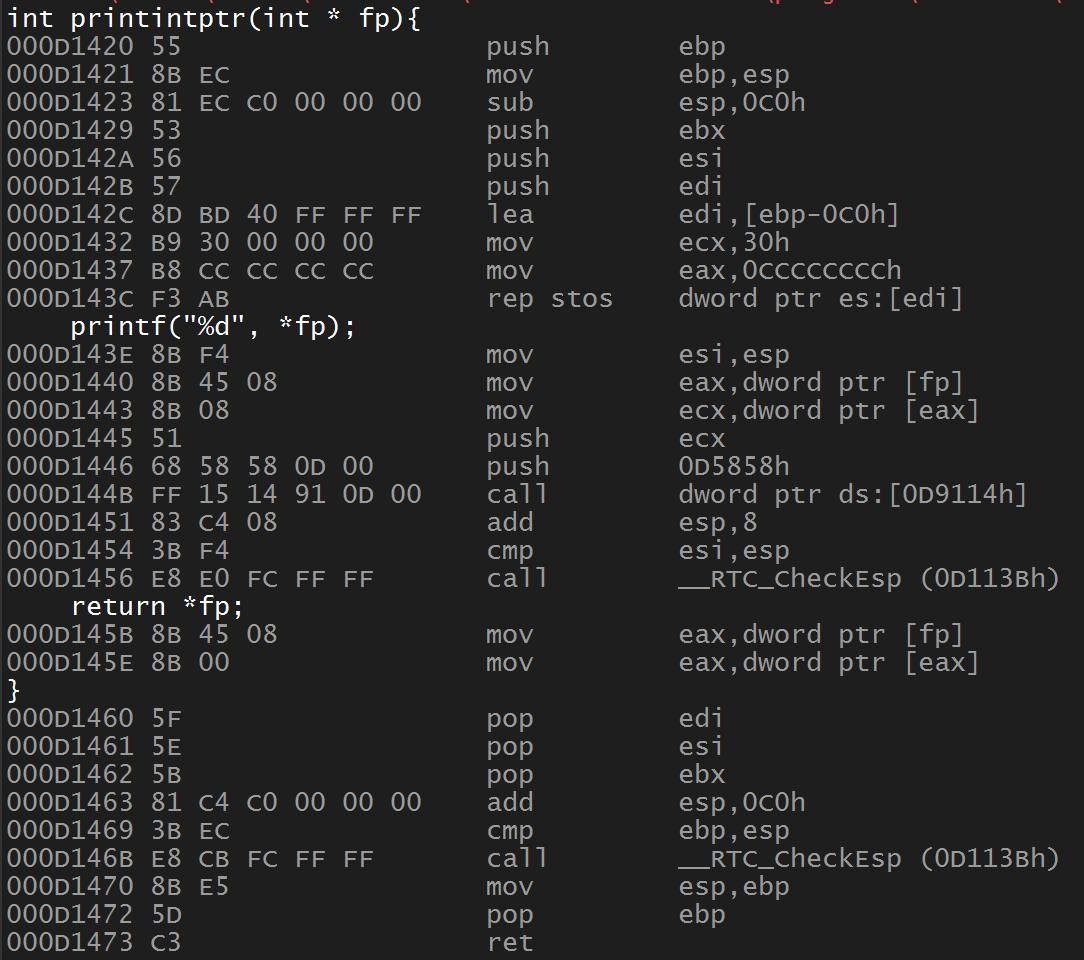
* 执行函数体。
* 准备返回值

返回值会保存在eax中返回，如果返回值位数为8位，可以返回





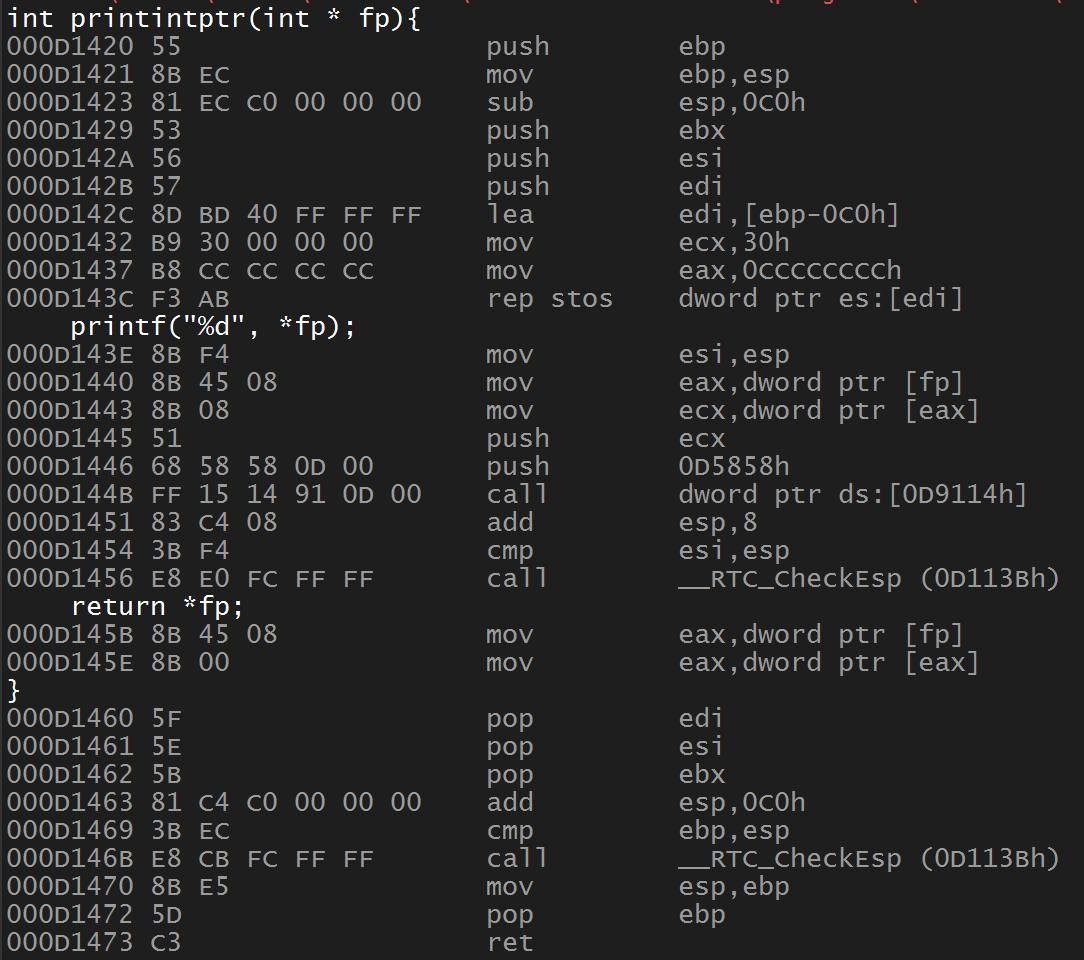
有返回值的返回值放在eax中返回。

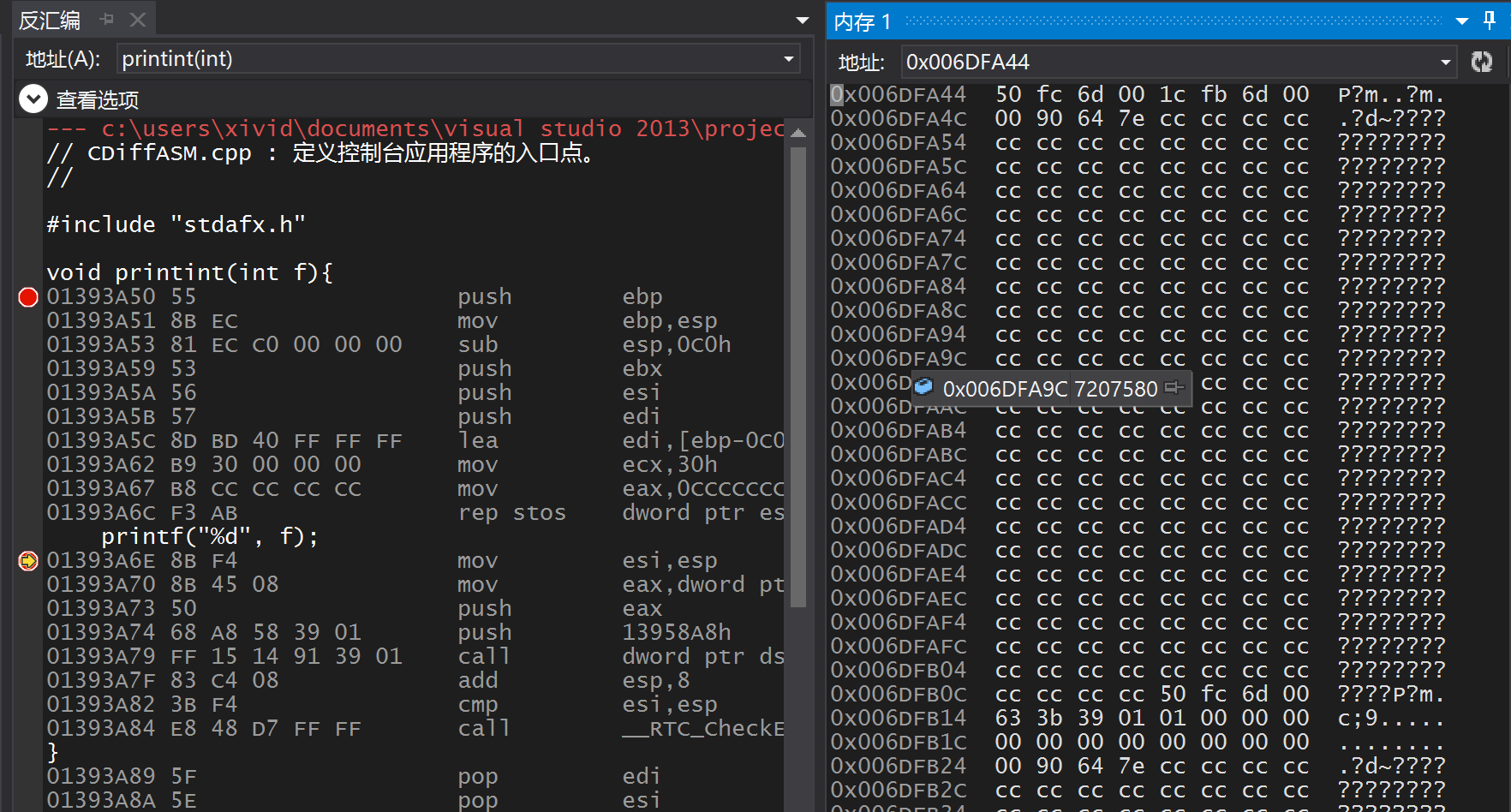


1. 堆栈框架

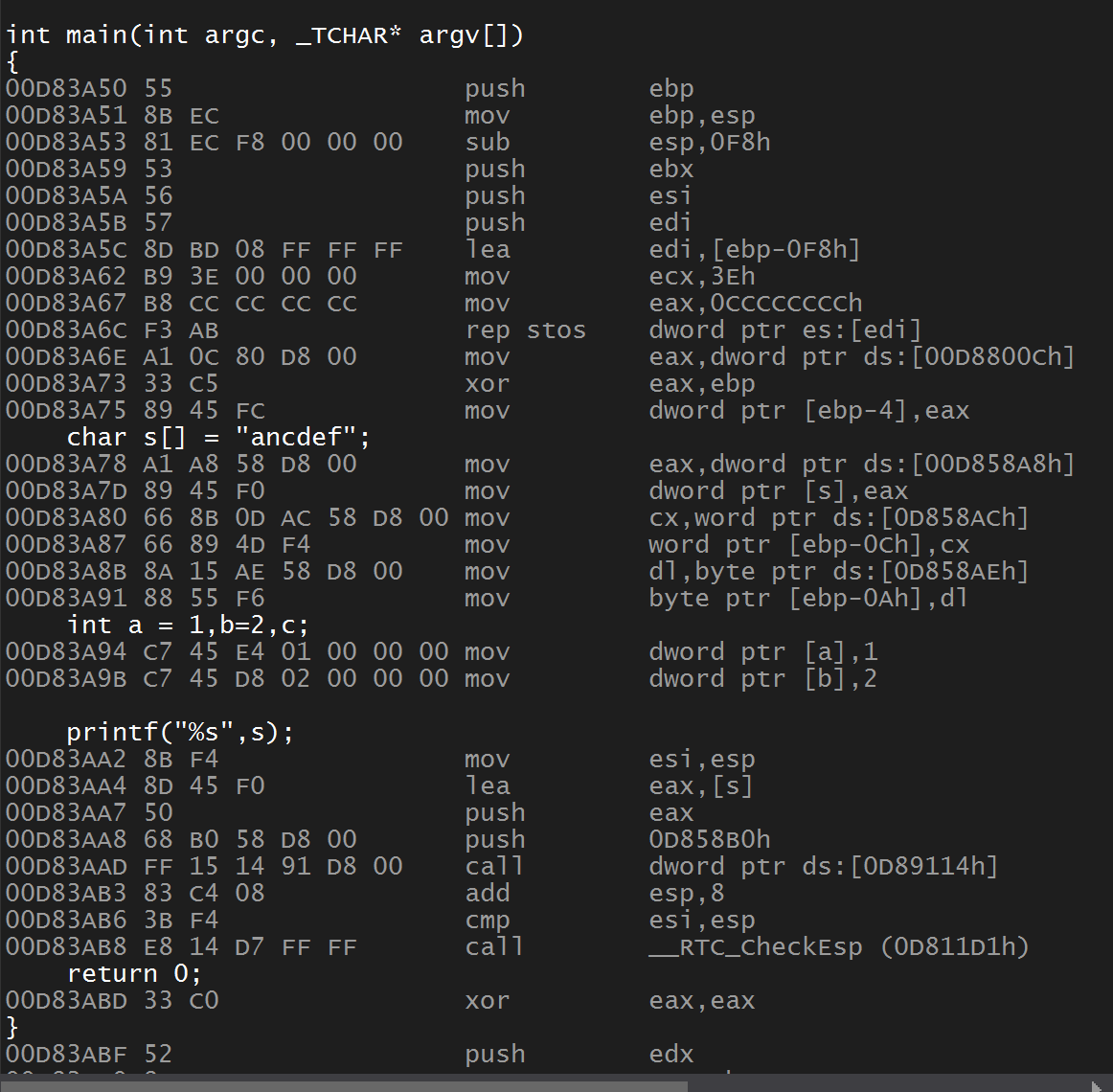
c语言编译器做出的堆栈框架与汇编语言略有不同。以下做出详细解释。

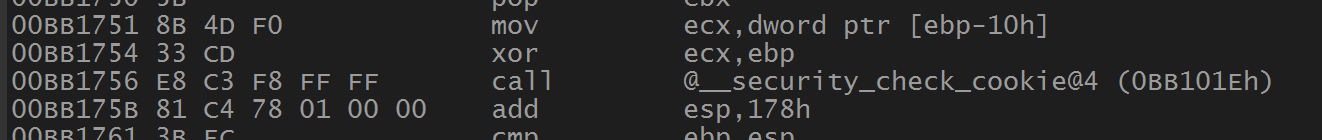
进入函数前需要先将函数参数逆序push入栈。然后调用call函数。返回地址入栈。原ebp入栈。保存现场，ebx esi edi 入栈。腾出局部变量空间使用cc填满，具体见函数一节。





特别的，主函数还要在完成以上操作之后堆栈压入esp^数据段地址。之后在退出主函数时使用此值^数据段地址，在比较esp与此异或结果，使用下图\_\_security\_check\_cookie\_\_方法检查异常。

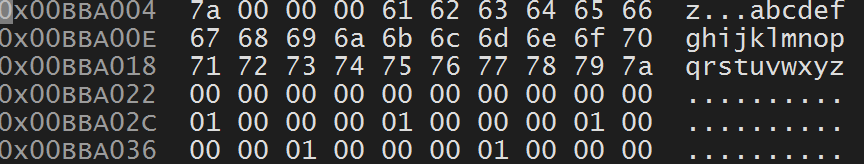




1. 全局、局部变量

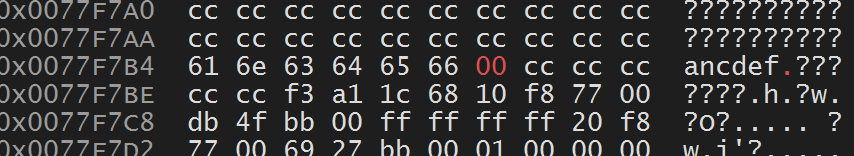
* 全局变量

C语言的全局变量保存在本章1节中描述的全局变量的保存位置处，保存方式是顺序存储的，这对应汇编中数据段（.data）的形式。与汇编程序有区别的地方是，我们在汇编中使用了宏定义方式（$char）使用常量字符串，这些字符串也被保存在了数据段，并且和数据段变量在内存上是挨着的，但是C语言找了其他的一块地方来存储这些常量字符串，如1节中所描述的放常量位置。



* 局部变量

C语言在每个函数（包括主函数）进入的时候都会分配一定的堆栈空间来存储局部变量，如本章4节中所说的使用cc cc … cc cc占满的区域。局部变量会以压栈方式，先声明的在大地址，后声明的在小地址，每个局部变量会用0补齐四字节，并于其他局部变量保持8字节分隔区域，这段区域内是函数开始时填入的cc cc … cc cc 。



1. 算数运算及表达式计算

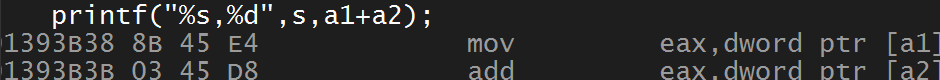
* 常量表达式

这种表达式会在编译的时候计算好，形成一个立即数，这个立即数是写在程序段的程序中的，在程序中使用mov add sub imul idiv等命令寄存器进行相关赋值或运算。

* 简单变量表达式计算

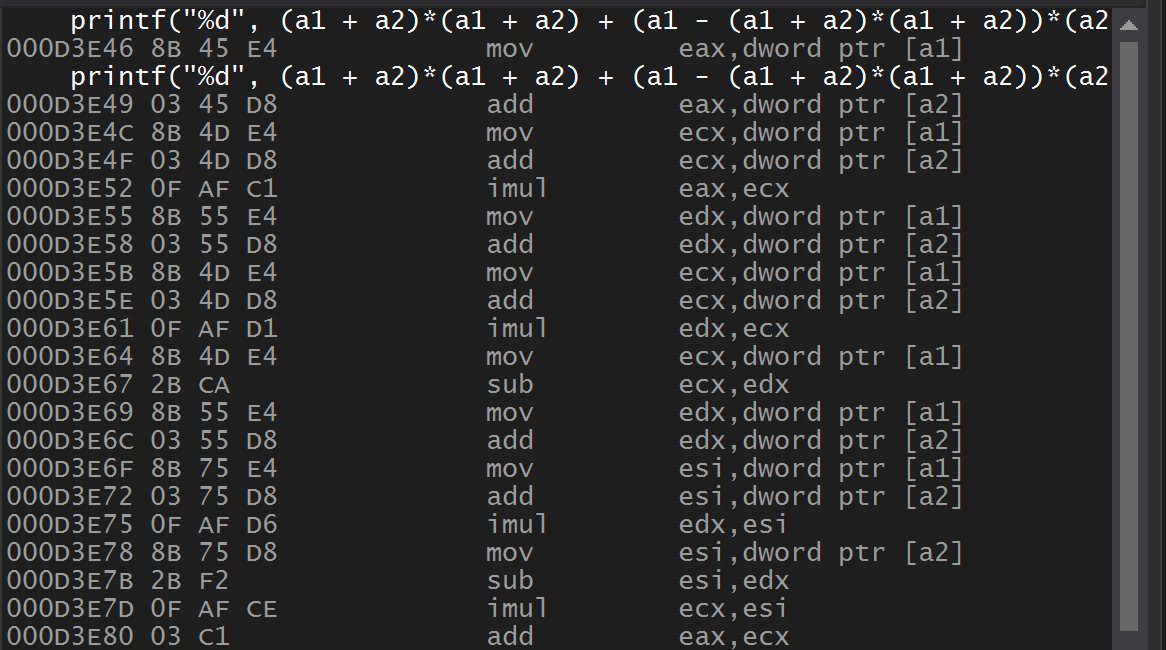
首先声明一下，这里说的表达式，不是纯变量常量表达式，期中也可以有常量，但是不会有可计算的常量表达式，如在变量表达式中出现可以事先计算的常量表达式，编译器也会在编译时计算成一个立即数。

如果一个表达式可以顺序计算，编译器会使用eax来存储每一步表达式的值，并使用mov（用于开始计算时赋值）、add、sub、imul、idiv命令进行运算，如果涉及long型等超过4位的数字计算，会使用edx来存储高位。



* 需要存储中间变量的表达式计算

如果出现计算(a1+a2)\*(a3+a4)等类似情况，就需要有多个寄存器来存储(a1+a2)和(a3+a4)的值，这时寄存器的使用顺序是eax，ecx，edx … 因为计算没有三元运算符，所以使用有限的寄存器就可以完成复杂计算，例如(a1+a2)\*(a3+a4)\*(a5+a6)并不需要三个寄存器，用eax存(a1+a2)、ecx存(a3+a4)之后eax \* ecx先得到前两项积存入eax，这样便可以再使用ecx。



1. C 程序编译、链接、装入内存

C语言代码采用编译方式翻译成机器码使得计算机能够执行。

* 编译

对C程序来说编译是对.c文件的操作，每个.c文件在最初都是单独编译的，所以，在每个C程序中都要说明其中引用函数、外部变量的来源，这些声明一般放在.h文件中，编译时只是确定使用的所有变量都有声明即可。

* 链接

c语言的链接就是对于一个c语言程序调用外部函数或者调用外部变量时，将外部内容的程序文件或二进制可执行文件链接如程序中。事实上这是一个将各种声明确定到一个定义（实现）上，将引用外部变量确定到这个变量的定义（那里分配了这个变量的地址）上的过程。

在这个过程中也负责给程序添加程序头、工程位置等等注释信息。

* 装入内存

c语言程序装入内存是操作系统在执行运行一个程序的指令时需要完成的。装入内存的是程序的机器码，既是本章第1节所说的除了堆栈段的其他部分，在这个过程中，完成了全局变量的初始化过程。

之后将IP指向程序开始位置，程序开始运行。

1. C 程序IO功能实现

C语言IO功能是调用系统或bios函数完成的，C语言由于其跨平台特性，并不能直接生成具有IO（IN\OUT）指令的代码。

第三章 c语言与汇编语言优缺点及适应场合

1. C语言优点，汇编语言的不足

* 可移植性，c语言相比汇编语言有较强的可移植性，在不同的硬件资源上都可以编译运行。
* 语法简单，学习成本低。适于编写一些大型项目。

1. 汇编语言优点，c语言的不足

* 汇编语言可以根据程序需要，提高运行效率和内存使用效率。比如之前记账中分析的，c语言在进入一个函数后需要腾出大量空间来存储局部变量，还要进行一些不必要的变量、寄存器等的保存，这样做时间空间上的消耗都很大。
* 汇编语言可以根据硬件的特点设计程序。C语言由于有可移植需求，无法充分使用硬件支持的所有指令来提高效率。比如交换两个数的算法，为了适应简单指令集的cpu，c语言并不能使用指令直接交换。
* 汇编语言可以进行一些硬件IO和扩展功能的实现。很多驱动需要汇编语言实现，这样才能完全发挥硬件功能。

1. c语言使用场合

* 不强调效率的场合。
* 不涉及直接对硬件操作的场合。在不同硬件平台没有什么区别，只是完成一些逻辑算法时可以使用。
* 需要跨平台的场合。
* 大型项目。因为c语言需要程序员考虑的硬件结构较少，可以让程序员更容易的开发大型项目，并且c语言具有可移植性，可以在多种平台上运行。

1. 汇编语言使用场合

* 效率要求苛刻的场合。如游戏引擎，物理引擎等。
* 对内存要求苛刻的场合，如嵌入式开发，单片机开发等。
* 要求直接操作硬件的场合。如硬件设备驱动等。
* 逆向工程。

第四章 课程收获及想法

1. 收获及想法

汇编语言是软件层面最底层的语言，是最接近机器的语言，学习这种语言可以更好地理解。

本课程学习完成，本人认为最大的收获就是了解并掌握了数字逻辑电路和高级语言代码软件之间中间层的秘密，使得对于计算机硬件和软件是如何联系起来的有了更深入的理解。

解决了原来一直想明白的问题，对硬件指令型编程方式的了解可以加深软件运行时的思考深入程度，可以再更深的层次上理解程序运行机制。

在之后的软、硬件学习中一定会由于汇编语言这一软硬件桥梁变得深入。

1. 一些建议

希望能够增加软硬件接触层部分，更偏向于硬件的学习，希望能够简单了解硬件是如何执行这些指令的。有关一些高级伪指令if,else等，希望能够减少，这些指令更接近于高级语言，个人感觉并不是汇编系统应该提倡的。